

TECH CHALLENGE – FIAP

Projeto: API para serviço de Reserva e Avaliação com Comentários de Restaurantes;

Curso: Arquitetura e Desenvolvimento em JAVA – Fase 3;

Escola: FIAP;

Grupo:

Victor Luiz Montibeller;

Leonardo Arantes Di Nizo;

Alexandre Marques;

Ferramentas utilizadas:

Event Storming: Miro;

Linguagem: JAVA 17;

Framework: Spring Boot, Spring JPA, Hibernate, Lombok;

Repositório: GitHub;

Banco de dados: H2 DataBase;

Publicação: Docker, Render;

Testes: JUnit, Mockito;

Dependências: Maven;

Sumário

Introdução	3
Contexto do Projeto	3
Importância da API	3
Objetivos do Projeto	3
Visão Geral do Projeto	4
Funcionalidades Principais	4
Tecnologias Utilizadas	4
Integração com Outros Serviços	5
Arquitetura do Software	6
Padrão MVC (Model-View-Controller)	6
Clean Architecture	6
Benefícios da Arquitetura	7
Qualidade de Software	8
Testes Unitários com JUnit e Mockito	8
Testes de Integração	8
Inspeção de Código	8
Cobertura de Testes com Coverage	9
Event Storming	10
Melhorias Futuras	11
Acesso ao Projeto	12
Configurando a API	13
Clonar o Repositório do GitHub:	13
Criar e Executar uma Imagem Docker:	13
Utilizar uma imagem Docker pública, no Registry do Docker:	13
Utilizar um Deploy Publicado em uma Plataforma Gratuita:	13
Testando a API	14
Endereços:	14
Clientes	15
Restaurantes:	16
Comentários:	17
Considerações Finais	19

Introdução

O Sistema de Reserva e Avaliação de Restaurantes é uma aplicação web desenvolvida em Java, projetada para atender às necessidades dos clientes e proprietários de restaurantes em um mundo cada vez mais digitalizado. Com o aumento do uso de dispositivos móveis e da internet, os consumidores esperam ter acesso rápido e fácil às informações sobre os restaurantes, bem como a capacidade de fazer reservas e compartilhar suas experiências online.

Contexto do Projeto

Nos dias de hoje, muitos consumidores confiam em plataformas online para encontrar e avaliar restaurantes antes de fazer uma reserva. Isso inclui a leitura de comentários de outros clientes, visualização de fotos do ambiente e do cardápio, e verificação da disponibilidade de mesas em tempo real. Por outro lado, os proprietários de restaurantes buscam maneiras de atrair mais clientes e melhorar a experiência dos que já frequentam seus estabelecimentos.

Importância da API

A API do Sistema de Reserva e Avaliação de Restaurantes visa preencher essa lacuna, proporcionando uma plataforma centralizada onde clientes e restaurantes podem interagir de forma eficiente e conveniente. Ao oferecer recursos como reserva de mesas online, avaliações e comentários, a API ajuda a melhorar a experiência do usuário e a promover a transparência e a confiança entre os consumidores e os estabelecimentos gastronômicos.

Objetivos do Projeto

Os principais objetivos do projeto incluem:

- Facilitar a reserva de mesas em restaurantes, oferecendo aos clientes uma maneira rápida e conveniente de encontrar e reservar um lugar para comer.
- Permitir que os clientes avaliem sua experiência em restaurantes e compartilhem feedbacks úteis para outros consumidores.
- Auxiliar os proprietários de restaurantes a gerenciar suas operações de forma mais eficiente, permitindo o acompanhamento das reservas, a análise das avaliações e o aprimoramento dos serviços.

Visão Geral do Projeto

O Sistema de Reserva e Avaliação de Restaurantes é uma aplicação web desenvolvida em Java, utilizando o framework Spring Boot. A API foi projetada para fornecer funcionalidades abrangentes para gerenciar reservas em restaurantes, permitindo aos usuários avaliar suas experiências e deixar comentários sobre os estabelecimentos visitados.

Funcionalidades Principais

A API oferece os seguintes recursos principais:

- **Cadastro de Clientes:** Permite que os clientes se cadastrem na plataforma fornecendo informações como nome, e-mail e data de cadastro.
- **Cadastro de Endereços:** Permite o cadastro de endereços dos clientes e dos restaurantes, incluindo informações como rua, número, bairro, cidade, estado, país e CEP.
- **Cadastro de Restaurantes:** Permite que os restaurantes sejam registrados na plataforma, fornecendo detalhes como nome, tipo de cozinha, horário de funcionamento, capacidade de assentos etc.
- **Gerenciamento de Reservas:** Os clientes podem fazer reservas em restaurantes disponíveis, especificando a data, hora, número de pessoas e preferências de mesa. Os restaurantes podem visualizar e confirmar as reservas pendentes.
- **Avaliação e Comentários:** Após a visita ao restaurante, os clientes têm a oportunidade de avaliar sua experiência e deixar comentários sobre a comida, serviço, ambiente etc.

Tecnologias Utilizadas

A API foi desenvolvida utilizando as seguintes tecnologias e ferramentas:

- **Java:** Linguagem de programação principal para o desenvolvimento da aplicação.
- **Spring Boot:** Framework Java usado para criar aplicativos baseados em Spring com facilidade.
- **Hibernate:** Biblioteca ORM para mapeamento objeto-relacional e persistência de dados.
- **Docker:** Conjunto de produtos de plataforma como serviço (PaaS) que usam virtualização de nível de sistema operacional para entregar software em pacotes chamados contêineres.
- **Lombok:** Framework para Java que permite escrever código eliminando a verbosidade, o que permite ganhar tempo de desenvolvimento para o que realmente é importante.

- H2: Sistema de gerenciamento de banco de dados relacional (RDBMS) escrito em Java.
- Spring Data JPA: Implementação da camada de acesso a dados baseada em JPA.
- Spring MVC: Padrão de projeto usado para desenvolver aplicativos web em Java.
- Maven: Ferramenta desenvolvida pela Apache, ela serve para gerenciar as dependências e automatizar seus builds.
- JUnit e Mockito: Frameworks de teste unitário e mocking para garantir a qualidade do software.
- Swagger: Ferramenta para projetar, construir, documentar e consumir serviços da Web RESTful.

Integração com Outros Serviços

A API pode ser integrada com outros serviços externos, como sistemas de pagamento, serviços de geolocalização, sistemas de notificação etc., para estender suas funcionalidades e melhorar a experiência do usuário.

Arquitetura do Software

A arquitetura do software é uma parte fundamental do projeto, pois define a estrutura geral da aplicação e como suas diferentes partes se relacionam entre si. No caso do Sistema de Reserva e Avaliação de Restaurantes, utilizamos uma combinação dos padrões MVC e Clean Architecture para garantir uma arquitetura sólida e escalável.

Padrão MVC (Model-View-Controller)

O padrão MVC é amplamente utilizado em aplicações web para separar as preocupações de apresentação, lógica de negócio e acesso a dados. Nossa implementação do MVC no projeto é a seguinte:

- **Model (Modelo):** Representa a camada de negócios da aplicação, incluindo entidades como Cliente, Endereço, Restaurante etc. Essas entidades encapsulam os dados da aplicação e a lógica de negócio associada a eles.
- **View (Visão):** Responsável pela apresentação dos dados aos usuários. No contexto da API, a View é representada pelos endpoints RESTful que expõem os recursos da aplicação aos clientes.
- **Controller (Controlador):** Atua como intermediário entre a View e o Modelo, recebendo as requisições dos clientes, chamando a lógica de negócio apropriada e retornando as respostas correspondentes. Os controllers são responsáveis por rotear as requisições para as ações corretas e garantir uma interação suave entre a camada de apresentação e a camada de negócio.

Clean Architecture

A arquitetura do software foi projetada com base nos princípios da Clean Architecture, que enfatiza a separação de preocupações e a independência de frameworks externos. Nossa implementação da Clean Architecture inclui as seguintes camadas:

- A camada de Entities é o núcleo da aplicação e contém as entidades de domínio que representam os conceitos fundamentais do negócio. As entidades encapsulam os dados e o comportamento relacionados a esses conceitos e são independentes de qualquer framework ou tecnologia externa. Exemplos de entidades incluem Cliente, Endereço, Restaurante, Reserva etc.
- A camada de Controllers é responsável por lidar com as requisições HTTP, roteando-as para os serviços apropriados e retornando as respostas correspondentes aos clientes. Os controllers servem como a interface entre a aplicação e o mundo exterior e são responsáveis por receber, validar e encaminhar as requisições para os serviços adequados. Eles são implementados de forma independente dos detalhes de implementação dos serviços e das entidades de domínio.

- A camada de Services contém a lógica de negócio da aplicação, representada por serviços que implementam os casos de uso específicos do sistema. Os serviços encapsulam as operações que podem ser realizadas na aplicação e são responsáveis por coordenar a interação entre as entidades de domínio e os repositórios de dados. Eles são independentes dos detalhes de implementação dos controllers e dos repositórios de dados.
- A camada de Repository é responsável pelo acesso e persistência dos dados no banco de dados. Os repositórios fornecem uma abstração sobre o armazenamento de dados e permitem que os serviços acessem e manipulem as entidades de domínio de forma transparente. Eles encapsulam as operações de consulta e persistência e são implementados de forma independente dos detalhes de implementação dos serviços e das entidades de domínio.

Benefícios da Arquitetura

A arquitetura do software baseada nos princípios da Clean Architecture oferece os seguintes benefícios:

- Separação de Preocupações: As responsabilidades de apresentação, lógica de negócio e acesso a dados são claramente separadas, facilitando a manutenção e a evolução da aplicação:
- Independência de Tecnologia: As camadas internas da aplicação são independentes de frameworks externos e detalhes de implementação, permitindo uma maior flexibilidade e adaptabilidade à medida que os requisitos do sistema mudam.
- Testabilidade: A arquitetura limpa facilita a escrita de testes automatizados, pois as diferentes partes do sistema podem ser testadas de forma isolada e sem depender de infraestrutura externa.

Qualidade de Software

A qualidade de software desempenha um papel fundamental na confiabilidade, segurança e eficácia da nossa API de Reservas e Avaliação de Restaurantes. Para garantir altos padrões de qualidade em nosso projeto, adotamos diversas práticas e ferramentas que promovem a detecção precoce de defeitos e a melhoria contínua do código.

Testes Unitários com JUnit e Mockito

Os testes unitários são essenciais para garantir que cada componente da nossa API funcione conforme o esperado. Utilizamos o framework JUnit para escrever e executar testes unitários de forma automatizada. Isso nos permite verificar o comportamento de métodos e classes individuais, garantindo que cada unidade de código atenda aos requisitos especificados.

Além disso, empregamos o Mockito para criar mocks de objetos durante os testes unitários. Essa prática nos permite isolar as unidades de código em teste e simular o comportamento de dependências externas, aumentando a eficiência e a confiabilidade dos testes.

Testes de Integração

Os testes de integração são cruciais para validar a interação entre os diferentes componentes da nossa API. Implementamos testes de integração para garantir que os diversos módulos da aplicação se integrem corretamente e produzam o resultado esperado. Esses testes são executados em um ambiente que replica o ambiente de produção, garantindo uma validação realista das interações entre os componentes.

Inspeção de Código

A inspeção de código é uma prática indispensável para identificar e corrigir potenciais problemas de qualidade no código fonte. Realizamos revisões de código periódicas para identificar e corrigir problemas de estilo, complexidade excessiva, bugs e outras questões de qualidade. Essa atividade colaborativa envolve membros da equipe de desenvolvimento que revisam o código uns dos outros em busca de possíveis melhorias e correções.

Cobertura de Testes com Coverage

A cobertura de testes é uma métrica importante que indica a porcentagem do código fonte que é exercida por testes automatizados. Utilizamos ferramentas de análise de cobertura, como JaCoCo, para avaliar a cobertura de testes da nossa API. Essas ferramentas nos fornecem insights sobre quais partes do código não estão sendo testadas adequadamente, ajudando a identificar lacunas na cobertura de testes que precisam ser abordadas.

Para executar os testes com cobertura, execute os comandos abaixo:

- mvn clean test;
- mvn jacoco:report;

Dentro do diretório target/site/jacoco/, você encontrará os relatórios HTML detalhando a cobertura de cada classe em seu projeto. Esses relatórios mostram a porcentagem de linhas de código cobertas por testes.

ReservasRestaurantes												
ReservasRestaurantes												
Element	Missed Instructions	Cov	Missed Branches	Cov	Missed Cxty	Missed Lines	Missed Methods	Missed Classes				
com.fiap.ReservasRestaurantes.restaurante.service		1%		0%	21 22	73 74	13 14	0 1				
com.fiap.ReservasRestaurantes.reserva.service		1%		0%	22 23	65 66	18 19	0 1				
com.fiap.ReservasRestaurantes.mesa.service		1%		0%	15 16	51 52	11 12	0 1				
com.fiap.ReservasRestaurantes.cliente.service		1%		0%	17 18	47 48	14 15	0 1				
com.fiap.ReservasRestaurantes.endereco.service		2%		0%	9 10	40 41	8 9	0 1				
com.fiap.ReservasRestaurantes.comentario.service		2%		0%	9 10	38 39	8 9	0 1				
com.fiap.ReservasRestaurantes.horario.service		2%		0%	9 10	37 38	8 9	0 1				
com.fiap.ReservasRestaurantes.reserva.controller		10%		n/a	6 8	17 21	6 8	0 1				
com.fiap.ReservasRestaurantes.restaurante.controller		10%		n/a	10 12	13 17	10 12	0 1				
com.fiap.ReservasRestaurantes.reserva.entity		36%		n/a	21 24	11 14	21 24	0 1				
com.fiap.ReservasRestaurantes.mesa.controller		11%		n/a	6 8	17 21	6 8	0 1				
com.fiap.ReservasRestaurantes.restaurante.DTO		0%		n/a	12 12	1 1	12 12	1 1				
com.fiap.ReservasRestaurantes.reserva.DTO		0%		n/a	12 12	1 1	12 12	1 1				
com.fiap.ReservasRestaurantes.restaurante.entity		41%		n/a	19 24	9 14	19 24	0 1				
com.fiap.ReservasRestaurantes.excecoes		4%		n/a	7 8	14 15	7 8	2 3				
com.fiap.ReservasRestaurantes.cliente.controller		14%		n/a	7 9	10 14	7 9	0 1				
com.fiap.ReservasRestaurantes.endereco.entity		38%		n/a	15 18	8 11	15 18	0 1				
com.fiap.ReservasRestaurantes.horario.controller		15%		n/a	4 6	11 15	4 6	0 1				
com.fiap.ReservasRestaurantes.endereco.DTO		0%		n/a	9 9	1 1	9 9	1 1				
com.fiap.ReservasRestaurantes.horario.entity		35%		n/a	14 16	8 10	14 16	0 1				
com.fiap.ReservasRestaurantes.horario.DTO		0%		n/a	8 8	1 1	8 8	1 1				
com.fiap.ReservasRestaurantes.mesa.entity		43%		n/a	12 16	6 10	12 16	0 1				
com.fiap.ReservasRestaurantes.comentario.DTO		0%		n/a	7 7	1 1	7 7	1 1				
com.fiap.ReservasRestaurantes.mesa.DTO		0%		n/a	7 7	1 1	7 7	1 1				
com.fiap.ReservasRestaurantes.endereco.controller		20%		n/a	4 6	7 11	4 6	0 1				
com.fiap.ReservasRestaurantes.comentario.controller		20%		n/a	4 6	7 11	4 6	0 1				
com.fiap.ReservasRestaurantes.comentario.entity		45%		n/a	6 14	6 9	6 14	0 1				
com.fiap.ReservasRestaurantes.cliente.DTO		0%		n/a	6 6	1 1	6 6	1 1				
com.fiap.ReservasRestaurantes.cliente.entity		89%		n/a	2 14	1 9	2 14	0 1				
com.fiap.ReservasRestaurantes.restaurante.entity.enumerations		95%		n/a	2 6	2 24	2 6	0 2				
com.fiap.ReservasRestaurantes.horario.entity.enumerations		93%		n/a	2 6	2 20	2 6	0 2				
com.fiap.ReservasRestaurantes.mesa.entity.enumerations		90%		n/a	2 6	2 15	2 6	0 2				
com.fiap.ReservasRestaurantes		37%		n/a	1 2	2 3	1 2	0 1				
com.fiap.ReservasRestaurantes.reserva.entity.enumerations		91%		n/a	1 3	1 8	1 3	0 1				
Total	2,463 of 3,133	21%	44 of 44	0%	308 382	512 637	286 360	9 39				

Event Storming

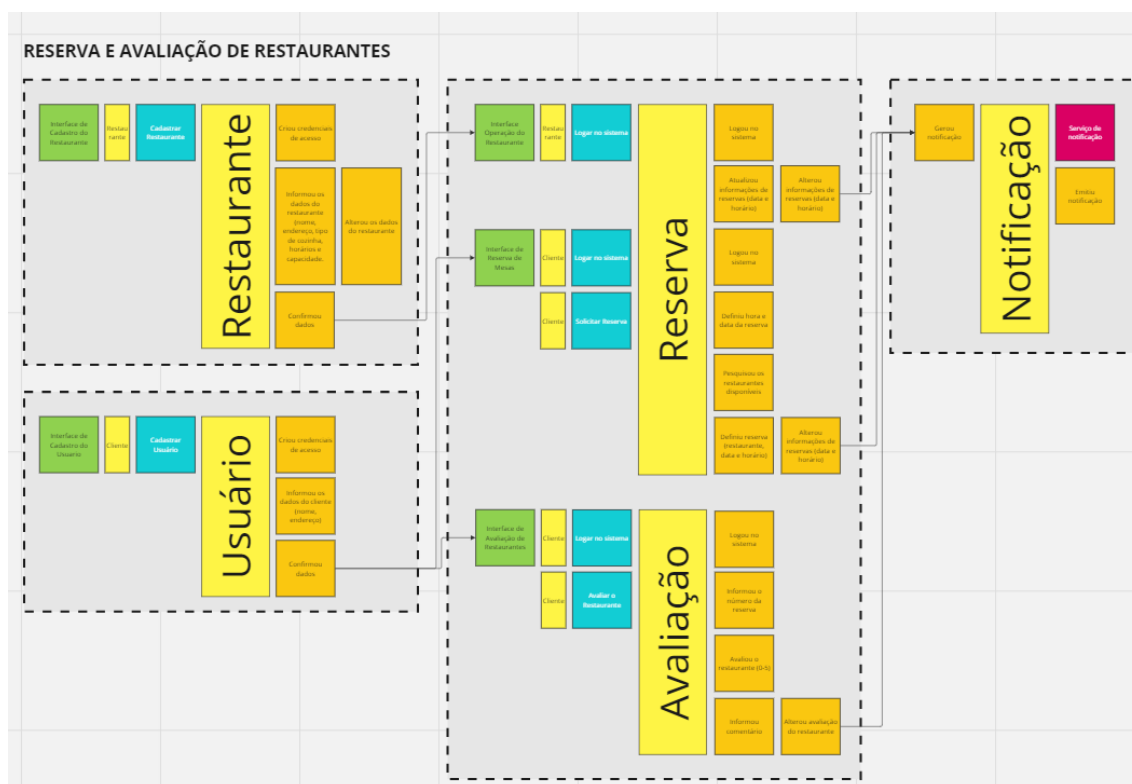
Iniciando com um brainstorming, discutindo as ideias do projeto entre os participantes do grupo que descrevessem os eventos e gatilhos da nossa API, simulando uma reunião com os especialistas e solicitantes do negócio.

Organizamos os comandos, atores, interfaces, políticas do negócio e eventos pivotais, identificando onde inicia e onde termina cada parte do processo.

Em seguida aplicamos o objetivo principal, agregados e contextos, separando-os para melhor visualização.

Em cada fase, novas ideias foram surgindo e assim fomos refinando o processo ao longo de sua construção. Todo esse processo de desenvolvimento do Event Storming e suas fases, podem ser visualizados através do link abaixo:

https://miro.com/app/board/uXiVNqrQfdY=



Melhorias Futuras

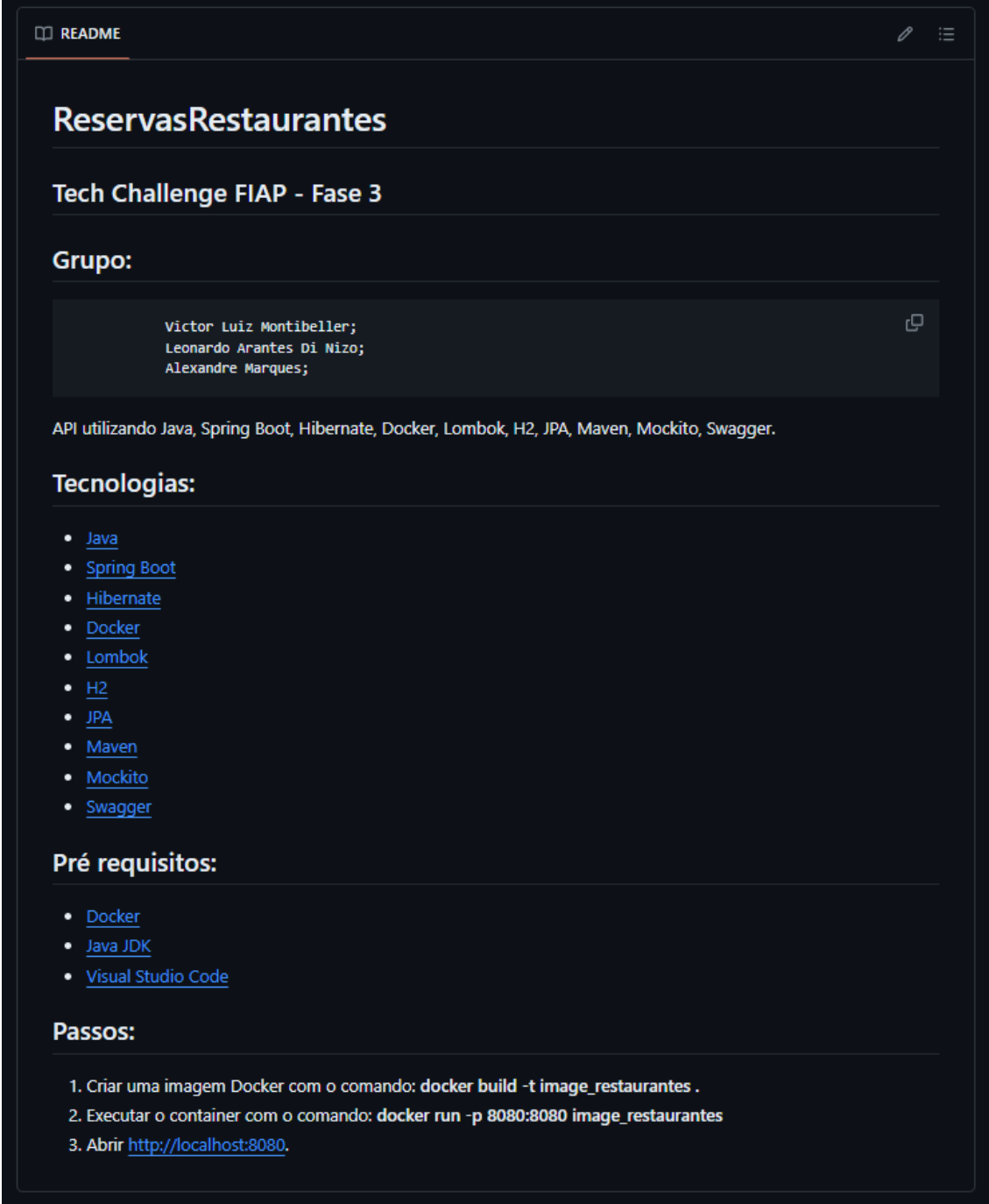
Após a entrega do projeto funcionando com os requisitos solicitados, deixamos o planejamento para algumas melhorias futuras, como:

- Implementação de Autenticação e Autorização: Adicionar autenticação e autorização para proteger endpoints sensíveis da API, garantindo que apenas usuários autorizados possam acessá-los.
- Melhorias na Segurança: Implementar práticas de segurança adicionais, como proteção contra ataques de injeção SQL, XSS e CSRF, além de configurar HTTPS para comunicação segura.
- Melhorias de Desempenho: Realizar otimizações de consulta e indexação no banco de dados para melhorar o desempenho da API, especialmente em cenários de alta carga.
- Implementação de Cache: Utilizar técnicas de cache para armazenar em cache dados frequentemente acessados e reduzir a carga no banco de dados, melhorando a escalabilidade e o desempenho geral da aplicação.
- Aprimoramento da Documentação da API: Refinar e expandir a documentação da API para torná-la mais abrangente e amigável para os desenvolvedores, incluindo exemplos de solicitações e respostas.
- Implementação de Testes de Integração Completa: Desenvolver testes de integração completos que cobrem todos os casos de uso principais da aplicação, garantindo uma cobertura de teste abrangente. problemas em tempo real e tomar medidas corretivas proativas.
- Adição de Recursos Adicionais: Considerar a adição de novos recursos, como suporte a múltiplos idiomas, integração com serviços de pagamento online para reservas pagas, ou funcionalidades de recomendação personalizada de restaurantes.
- Interface gráfica: Front-End para integração com a API, possibilitando uma melhor adaptação aos processos, como um app WEB e versão MóBILE.
- Notificações: Enviar lembretes de notificações por SMS e e-mail, com a possibilidade de cadastrar o evento automaticamente na agenda dos usuários.

Acesso ao Projeto

O projeto da API para serviço de Reserva e Avaliação com Comentários de Restaurantes, codificado seguindo o que foi definido no Objetivo do Projeto, está disponível no repositório do GitHub:

<https://github.com/victormontibeller/ReservasRestaurantes>



The image is a screenshot of a GitHub repository's README file for a project named 'ReservasRestaurantes'. The interface is in dark mode. At the top, there's a 'README' tab and some icons. The title 'ReservasRestaurantes' is prominently displayed. Below it, the subtitle 'Tech Challenge FIAP - Fase 3' is shown. A section titled 'Grupo:' lists the team members: Victor Luiz Montibeller, Leonardo Arantes Di Nizo, and Alexandre Marques. Below this, a line of text states the API uses Java, Spring Boot, Hibernate, Docker, Lombok, H2, JPA, Maven, Mockito, and Swagger. A 'Tecnologias:' section follows, listing these technologies as bullet points with links. Then, a 'Pré requisitos:' section lists Docker, Java JDK, and Visual Studio Code as bullet points with links. Finally, a 'Passos:' section provides three numbered steps: 1. Create a Docker image with 'docker build -t image_restaurantes .', 2. Run the container with 'docker run -p 8080:8080 image_restaurantes', and 3. Open 'http://localhost:8080'.

README

ReservasRestaurantes

Tech Challenge FIAP - Fase 3

Grupo:

Victor Luiz Montibeller;
Leonardo Arantes Di Nizo;
Alexandre Marques;

API utilizando Java, Spring Boot, Hibernate, Docker, Lombok, H2, JPA, Maven, Mockito, Swagger.

Tecnologias:

- [Java](#)
- [Spring Boot](#)
- [Hibernate](#)
- [Docker](#)
- [Lombok](#)
- [H2](#)
- [JPA](#)
- [Maven](#)
- [Mockito](#)
- [Swagger](#)

Pré requisitos:

- [Docker](#)
- [Java JDK](#)
- [Visual Studio Code](#)

Passos:

1. Criar uma imagem Docker com o comando: `docker build -t image_restaurantes .`
2. Executar o container com o comando: `docker run -p 8080:8080 image_restaurantes`
3. Abrir <http://localhost:8080>.

Configurando a API

Clonar o Repositório do GitHub:

Para configurar a API, basta clonar o projeto:

<https://github.com/victormontibeller/ReservasRestaurantes>

Criar e Executar uma Imagem Docker:

- No terminal, construa a imagem Docker usando o seguinte comando:
 - **`docker build -t image_restaurantes .`**
- Após a construção da imagem, execute um contêiner Docker com o seguinte comando:
 - **`docker run -p 8080:8080 image_restaurantes`**
- Abrir a URL abaixo:
 - <http://localhost:8080>

Utilizar uma imagem Docker pública, no Registry do Docker:

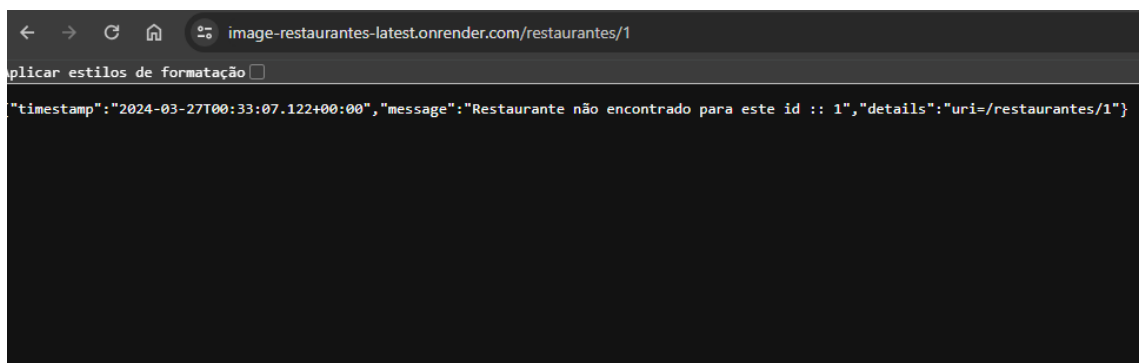
Acessar a imagem através da URL abaixo:

- [Docker.io/victormontibeller/image_restaurantes:latest](https://victormontibeller/image_restaurantes:latest);

Utilizar um Deploy Publicado em uma Plataforma Gratuita:

Acessar a API pública através da URL abaixo:

- <https://image-restaurantes-latest.onrender.com/>

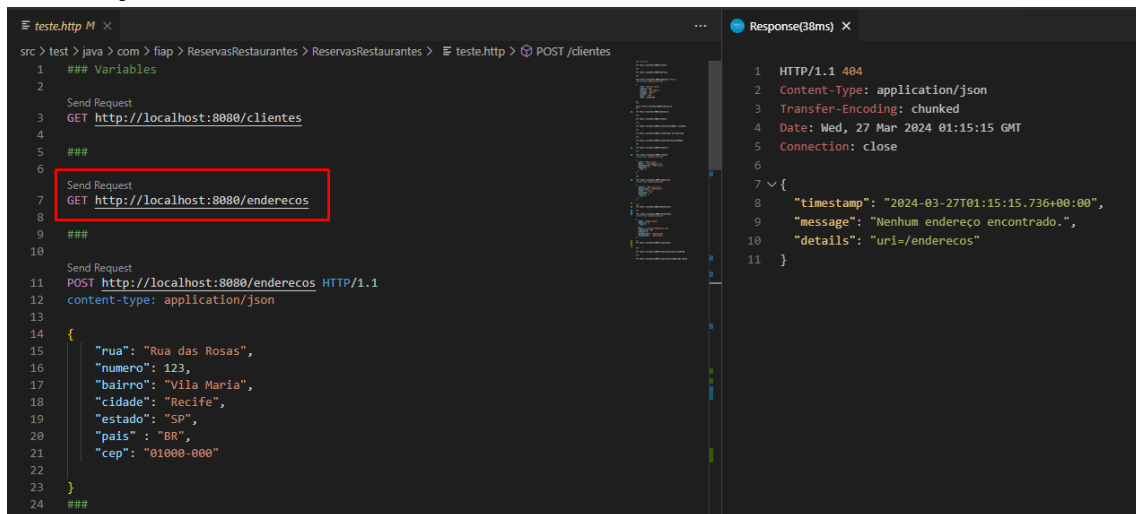


Testando a API

Para testar nossa API, temos alguns exemplos de payloads do tipo Json para o envio de requisições via Postman ou outra ferramenta semelhante. Esses payloads se encontram no projeto.

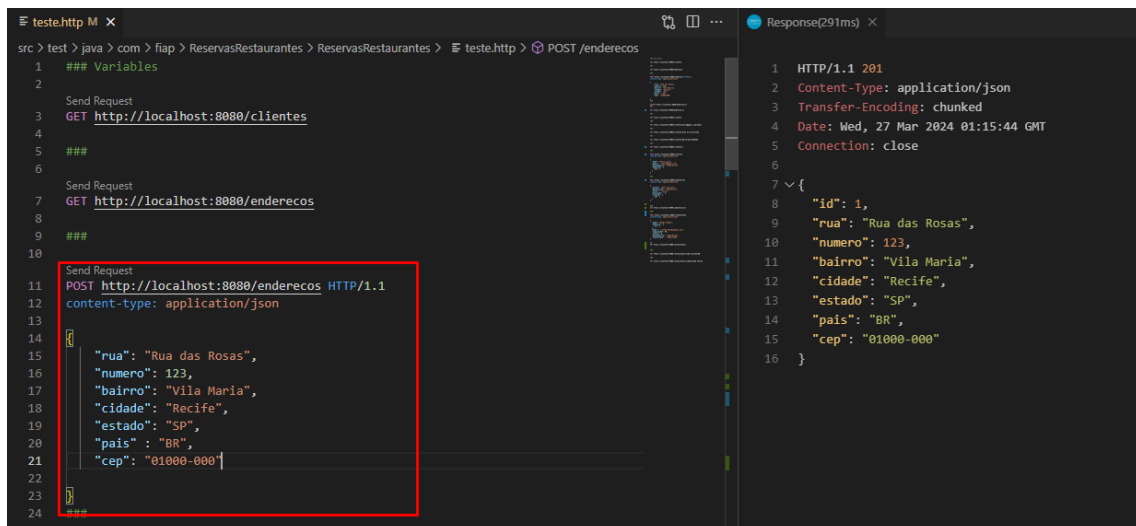
Necessário seguir o fluxo para que a aplicação tenha os dados necessários para gerar corretamente as Reservas.

Endereços:



```
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > POST /clientes
1  ### Variables
2
3  Send Request
4  GET http://localhost:8080/clientes
5
6  ###
7  Send Request
8  GET http://localhost:8080/enderecos
9
10 ###
11 Send Request
12 POST http://localhost:8080/enderecos HTTP/1.1
13 content-type: application/json
14 {
15   "rua": "Rua das Rosas",
16   "numero": 123,
17   "bairro": "Vila Maria",
18   "cidade": "Recife",
19   "estado": "SP",
20   "pais": "BR",
21   "cep": "01000-000"
22 }
23
24 ###
25
```

```
1 HTTP/1.1 404
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:15:15 GMT
5 Connection: close
6
7 {
8   "timestamp": "2024-03-27T01:15:15.736+00:00",
9   "message": "Nenhum endereço encontrado.",
10  "details": "uri=/enderecos"
11 }
```



```
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > POST /enderecos
1  ### Variables
2
3  Send Request
4  GET http://localhost:8080/clientes
5
6  ###
7  Send Request
8  GET http://localhost:8080/enderecos
9
10 ###
11 Send Request
12 POST http://localhost:8080/enderecos HTTP/1.1
13 content-type: application/json
14 {
15   "rua": "Rua das Rosas",
16   "numero": 123,
17   "bairro": "Vila Maria",
18   "cidade": "Recife",
19   "estado": "SP",
20   "pais": "BR",
21   "cep": "01000-000"
22 }
23
24 ###
25
```

```
1 HTTP/1.1 201
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:15:44 GMT
5 Connection: close
6
7 {
8   "id": 1,
9   "rua": "Rua das Rosas",
10  "numero": 123,
11  "bairro": "Vila Maria",
12  "cidade": "Recife",
13  "estado": "SP",
14  "pais": "BR",
15  "cep": "01000-000"
16 }
```

```
teste.http M x
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > ...
1  ## Variables
2
3  Send Request
4  GET http://localhost:8080/clientes
5
6  ##
7
8  Send Request
9  GET http://localhost:8080/enderecos
10
11  ##
12
13  Send Request
14  POST http://localhost:8080/enderecos HTTP/1.1
15  content-type: application/json
16
17  {
18    "rua": "Rua das Rosas",
19    "numero": 123,
20    "bairro": "Vila Maria",
21    "cidade": "Recife",
22    "estado": "SP",
23    "pais": "BR",
24    "cep": "01000-000"
25  }
26
27  ##
28
29  Send Request
30  DELETE http://localhost:8080/enderecos/1
31
32  ##
```

```
Response(34ms) x
1  HTTP/1.1 200
2  Content-Type: application/json
3  Transfer-Encoding: chunked
4  Date: Wed, 27 Mar 2024 01:17:05 GMT
5  Connection: close
6
7  [
8    {
9      "id": 1,
10     "rua": "Rua das Rosas",
11     "numero": 123,
12     "bairro": "Vila Maria",
13     "cidade": "Recife",
14     "estado": "SP",
15     "pais": "BR",
16     "cep": "01000-000"
17   }
18 ]
```

Clientes:

```
teste.http M x
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > ...
11 POST http://localhost:8080/enderecos HTTP/1.1
12
13 ##
14
15 Send Request
16 DELETE http://localhost:8080/enderecos/1
17
18 ##
19
20 Send Request
21 GET http://localhost:8080/enderecos/1
22
23 ##
24
25 Send Request
26 GET http://localhost:8080/clientes
27
28 ##
29
30 Send Request
31 GET http://localhost:8080/clientes/maria@gmail.com/email
32
33 ##
34
35 Send Request
36 GET http://localhost:8080/clientes/Alex da Silva/nome
37
38 ##
39
40 Send Request
41 GET http://localhost:8080/clientes/Maria/parteDoNome
```

```
Response(26ms) x
1  HTTP/1.1 404
2  Content-Type: application/json
3  Transfer-Encoding: chunked
4  Date: Wed, 27 Mar 2024 01:18:13 GMT
5  Connection: close
6
7  {
8    "timestamp": "2024-03-27T01:18:13.144+00:00",
9    "message": "Nenhum cliente encontrado.",
10    "details": "uri=/clientes"
11  }
```

```
teste.http M x
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > ...
46
47 ##
48
49 Send Request
50 GET http://localhost:8080/clientes/1
51
52 ##
53
54 Send Request
55 POST http://localhost:8080/clientes
56 content-type: application/json
57
58 {
59   "nome": "Maria Julia",
60   "email": "maria@gmail.com",
61   "dataCadastro": "2024-02-16",
62   "endereco": {
63     "id": 1
64   }
65 }
66
67 ##
68
69 Send Request
70 POST http://localhost:8080/comentarios
71 content-type: application/json
```

```
Response(122ms) x
1  HTTP/1.1 201
2  Content-Type: application/json
3  Transfer-Encoding: chunked
4  Date: Wed, 27 Mar 2024 01:18:31 GMT
5  Connection: close
6
7  {
8    "id": 1,
9    "nome": "Maria Julia",
10    "email": "maria@gmail.com",
11    "dataCadastro": "2024-02-16",
12    "endereco": {
13      "id": 1,
14      "rua": null,
15      "numero": 0,
16      "bairro": null,
17      "cidade": null,
18      "estado": null,
19      "pais": null,
20      "cep": null
21    }
22  }
```

```
teste.http M X
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > POST /clientes
40
Send Request
41 GET http://localhost:8080/clientes/Alex da Silva/nome
42
43 ###
44
Send Request
45 GET http://localhost:8080/clientes/Maria/parteDoNome
46
47 ###
48
Send Request
49 GET http://localhost:8080/clientes/1
50
51 ###
52
Send Request
53 POST http://localhost:8080/clientes
54 content-type: application/json
55
56 {
57   "nome": "Maria Julia",
58   "email": "maria@gmail.com",
59   "dataCadastro": "2024-02-16",
60   "endereco": {
61     "id": 1
62   }
63 }
```

```
Response(15ms) X
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:19:02 GMT
5 Connection: close
6
7 {
8   "id": 1,
9   "endereco": {
10     "id": 1,
11     "rua": "Rua das Rosas",
12     "numero": 123,
13     "bairro": "Vila Maria",
14     "cidade": "Recife",
15     "estado": "SP",
16     "pais": "BR",
17     "cep": "01000-000"
18   },
19   "reserva": [],
20   "nome": "Maria Julia",
21   "email": "maria@gmail.com",
22   "dataCadastro": "2024-02-16"
23 }
```

Restaurantes:

```
teste.http M X
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > GET /restaurantes/1
82 GET http://localhost:8080/comentarios/1
83
84 ###
85
Send Request
86 POST http://localhost:8080/restaurantes
87 content-type: application/json
88
89 {
90   "nome": "Manga Espada",
91   "endereco": {
92     "id": 1
93   },
94   "email": "restaurante@example.com",
95   "capacidade": 50,
96   "status": 0,
97   "dataCadastro": "2024-02-20",
98   "tipoCozinha": "BRASILEIRA"
99 }
100
101 ###
102
Send Request
103 GET http://localhost:8080/restaurantes/1
104
105 ###
106
Send Request
107 GET http://localhost:8080/restaurantes/cidade/Joaõ Pessoa
```

```
Response(42ms) X
1 HTTP/1.1 404
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:21:22 GMT
5 Connection: close
6
7 {
8   "timestamp": "2024-03-27T01:21:22.133+00:00",
9   "message": "Restaurante não encontrado para este id :: 1",
10   "details": "uri=/restaurantes/1"
11 }
```

```
teste.http M X
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > GET /restaurantes/1
80
81 ###
82 GET http://localhost:8080/comentarios/1
83
84 ###
85
Send Request
86 POST http://localhost:8080/restaurantes
87 content-type: application/json
88
89 {
90   "nome": "Manga Espada",
91   "endereco": {
92     "id": 1
93   },
94   "email": "restaurante@example.com",
95   "capacidade": 50,
96   "status": 0,
97   "dataCadastro": "2024-02-20",
98   "tipoCozinha": "BRASILEIRA"
99 }
100
101 ###
102
Send Request
103 GET http://localhost:8080/restaurantes/1
104
105 ###
106
Send Request
107 GET http://localhost:8080/restaurantes/cidade/Joaõ Pessoa
```

```
Response(56ms) X
1 HTTP/1.1 201
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:22:06 GMT
5 Connection: close
6
7 {
8   "id": 1,
9   "nome": "Manga Espada",
10   "endereco": {
11     "id": 1,
12     "rua": null,
13     "numero": 0,
14     "bairro": null,
15     "cidade": null,
16     "estado": null,
17     "pais": null,
18     "cep": null
19   },
20   "email": "restaurante@example.com",
21   "horario": null,
22   "reserva": null,
23   "mesa": null,
24   "tipoCozinha": "BRASILEIRA",
25   "capacidade": 50,
26   "status": "ATIVO",
27   "dataCadastro": "2024-02-20"
28 }
```



```
teste.http M
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > GET /restaurantes/1
81
###
Send Request
82 GET http://localhost:8080/comentarios/1
83
###
84
85
Send Request
86 POST http://localhost:8080/restaurantes
87 content-type: application/json
88
89 {
90   "nome": "Manga Espada",
91   "endereco": {
92     "id": 1
93   },
94   "email": "restaurante@example.com",
95   "capacidade": 50,
96   "status": 0,
97   "dataCadastro": "2024-02-20",
98   "tipoCozinha": "BRASILEIRA"
99 }
100
###
101
102
Send Request
103 GET http://localhost:8080/restaurantes/1
104
###
105
106
Send Request
107 GET http://localhost:8080/restaurantes/cidade/João Pessoa

Response(40ms) X
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:22:28 GMT
5 Connection: close
6
7 {
8   "id": 1,
9   "nome": "Manga Espada",
10  "endereco": {
11    "id": 1,
12    "rua": "Rua das Rosas",
13    "numero": 123,
14    "bairro": "Vila Maria",
15    "cidade": "Recife",
16    "estado": "SP",
17    "pais": "BR",
18    "cep": "01000-000"
19  },
20  "email": "restaurante@example.com",
21  "horario": [],
22  "reserva": [],
23  "mesa": [],
24  "tipoCozinha": "BRASILEIRA",
25  "capacidade": 50,
26  "status": "ATIVO",
27  "dataCadastro": "2024-02-20"
28 }
```

Comentários:

```
teste.http M
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > GET /comentarios/1
66
###
67
Send Request
68 POST http://localhost:8080/comentarios
69 content-type: application/json
70
71 {
72   "titulo": "Mané da Silva",
73   "dataCriacao": "2024-02-16",
74   "texto": "teste",
75   "avaliacao": 5,
76   "cliente": {
77     "id": 1
78   }
79 }
80
###
81
82
Send Request
83 GET http://localhost:8080/comentarios/1
84
###
85
86
Send Request
87 POST http://localhost:8080/restaurantes
88 content-type: application/json
89
90 {
91   "nome": "Manga Espada",
92   "endereco": {
93     "id": 1
94   },
95 }
```

```
teste.http M
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > GET /enderecos
63
}
64
###
65
66
Send Request
67 POST http://localhost:8080/comentarios
68 content-type: application/json
69
70 {
71   "titulo": "Mané da Silva",
72   "dataCriacao": "2024-02-16",
73   "texto": "teste",
74   "avaliacao": 5,
75   "cliente": {
76     "id": 1
77   }
78 }
79
###
80
81
Send Request
82 GET http://localhost:8080/comentarios/1
83
###
84
85
Send Request
86 POST http://localhost:8080/restaurantes
87 content-type: application/json
88
89 {
90   "nome": "Manga Espada",
91 }
```

```
teste.http M
src > test > java > com > fiap > ReservasRestaurantes > ReservasRestaurantes > teste.http > GET /comentarios/3
67 POST http://localhost:8080/comentarios
68 content-type: application/json
69
70 {
71   "titulo": "Mané da Silva",
72   "dataCriacao": "2024-02-16",
73   "texto": "teste",
74   "avaliacao": 5,
75   "cliente": {
76     "id": 1
77   }
78 }
79
80
81 ###
82 Send Request
83 GET http://localhost:8080/comentarios/3
84
85 ###
86 Send Request
87 POST http://localhost:8080/restaurantes
88 content-type: application/json
89
90 {
91   "nome": "Manga Espada",
92   "endereco": {
93     "id": 1
94   },
95   "email": "restaurante@example.com",
96   "capacidade": 50,
97   "status": 0,
98 }
```

```
Response(21ms) X
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Wed, 27 Mar 2024 01:29:16 GMT
5 Connection: close
6
7 {
8   "id": 3,
9   "cliente": {
10     "id": 1,
11     "endereco": {
12       "id": 1,
13       "rua": "Rua das Rosas",
14       "numero": 123,
15       "bairro": "Vila Maria",
16       "cidade": "Recife",
17       "estado": "SP",
18       "pais": "BR",
19       "cep": "01000-000"
20     },
21     "reserva": [],
22     "nome": "Maria Julia",
23     "email": "maria@gmail.com",
24     "dataCadastro": "2024-02-16"
25   },
26   "titulo": "Mané da Silva",
27   "texto": "teste",
28   "avaliacao": 5,
29   "dataCriacao": "2024-02-16"
```

Considerações Finais

O desenvolvimento da API de Reservas e Avaliação de Restaurantes representou uma jornada empolgante em direção à construção de uma solução robusta e de alta qualidade. Durante este projeto, demos destaque à arquitetura limpa, seguindo os princípios da Clean Architecture, que nos permitiram criar um código modular, escalável e de fácil manutenção.

Durante o desenvolvimento da API, exploramos diversas tecnologias e ferramentas, incluindo Spring, JPA, Hibernate e Mockito, e aplicamos práticas de engenharia de software, como testes unitários, testes de integração e análise de cobertura de testes.

Os testes desempenharam um papel crucial em nosso processo de desenvolvimento, com ênfase em testes unitários e de integração. Utilizamos ferramentas como JUnit e Mockito para garantir a qualidade e confiabilidade de nossa aplicação. Além disso, a integração com Docker facilitou a distribuição e execução da API em diferentes ambientes de desenvolvimento e produção.

Nossa jornada não foi apenas uma oportunidade de aplicar conceitos teóricos, mas também um aprendizado contínuo e colaborativo. A troca de conhecimentos e a resolução de desafios em equipe fortaleceram nossa compreensão e habilidades em engenharia de software.

À medida que concluímos este projeto, reconhecemos que sempre há espaço para melhorias e refinamentos. No entanto, estamos orgulhosos do que alcançamos e confiantes de que nossa API oferece uma base sólida para futuras iterações e extensões.

Agradecemos à FIAP por nos proporcionar esta oportunidade de aprendizado e crescimento, e aos nossos colegas e professores por seu apoio e orientação ao longo do curso. Estamos ansiosos para aplicar os conhecimentos adquiridos neste projeto em nossas jornadas futuras no desenvolvimento de software.